Ultimate Chess

Authors: Yoorae Kim, Demi Lee, Anoushka Tiwari: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system that lets a person play chess against an AI on a physical chess board without any software requirements. The system provides a simple interface to play chess against an AI opponent without having to use digital apps. The project is intended to help the elderly and people who may not be comfortable with technology enjoy the game. The game is a fast version of chess, with the AI making moves within 60 seconds. The players move is detected with 98% accuracy. The simplicity of the game is an improvement over state of the art digital chessboards that require complex app integration and hard to set up hardware.**

***Index Terms*— Background Subtraction, Canny, Chess, Computer Vision, Neural Network**

1 INTRODUCTION

Chess is one of the most popular games in the world. Over 600 million people worldwide enjoy the classic game. The elderly especially enjoy the game. According to some studies, it even helps prevent Alzheimer's and mental decline by stimulating the brain. However, the elderly are not always able to find someone to play with them. They are also often not as comfortable with technology to be able to enjoy a game against an Artificial Intelligence (AI) on a mobile app. Some feel that a mobile app simply doesn't provide the full chess experience.

Our smart chess board attempts to make chess more accessible to the elderly by providing a simple and easy-to-use way to play chess against an AI opponent. We provide a custom chess board that comes with a camera, LEDs and a push button. The user makes their move and presses the button. The LEDs at the initial and final square light up. If the user finds the move incorrect, they have 10 seconds to press the push button and remake their move 2 times. Then, the AI's move lights up. Once the user moves the AI's piece, the user presses the button the register this. After this, all LEDs light up indicating the user can make the next move. The requirements listed in this section are derived in the design requirements. The users require a high move detection accuracy (99%) and a low latency ($\leq$ 24 seconds). They also require high deviation tolerance, which means minor changes to a piece's location should not be detected as a move if the piece is still within the same square. This is measured as the maximum deviation which will not result in a change being detected in a square.

While there exist automatic chess boards in the market today, most of them require smartphone connectivity and need the player to use apps. Our target demographic is people who do not have access to a smartphone or simply don't want to use it. The simplicity of our product is an advantage over current smart boards. The limitations our approach is that the smoothness of the game is compromised because the user has to press buttons to indicate that the move is done.

2 DESIGN REQUIREMENTS

The goal of our project is to help people play chess in their free time. We therefore have to ensure that we have low latency because people may not have huge chunks of free time. We settled on a standard fast variant of chess called Fast chess. A popular type of fast chess gives people 2 minutes to make their move. Since people may be more patient playing against a person than waiting for a computer to make its move, we halved this time and made our timing requirement be 1 minute.

We thus have the following overall timing constraint:

$$tM + tL + tA \leq 60s \tag{1}$$

where tM is the time it takes for move detection (s), tL is the time it takes to determine if the move is legal (s) and tA is the time it takes to get and display the AI's next move from the AI engine (s).

The main areas of the project are Move detection, determining whether a move is legal, coming up with the AI move and displaying it on the LEDs. The requirements based on area are described below.

2.1 Move Detection

Detecting the player's move is a core part of the project. There are 3 quantitative requirements here: the detection time, the detection accuracy, and the number of tries it takes to come up with the correct detection. The detection time must meet the constraint described in equation (1).

2.1.1 Number of Retries

This requirement is due to the fact that we want to keep the human in the loop during the move detection to increase accuracy. When we finish detecting the move using the CV pipeline, we display the move the CV detected on the board by lighting up 2 LEDs (one for the initial position of the piece and one for the final). If this move is correct, the user does not need to do anything, and the detected move will go through the rest of the pipeline in 10 seconds. The reason we give the user 10 seconds here is due to the fact that our target demographic is the elderly, who often have a slow reaction time. If the move is wrong, the user has 10 seconds to press a button indicating this, and then make the move again. This helps mitigate issues with

the piece not being within the edges of the square or a bad image. We want the user to try 3 times before we end the game. The reason we settled on this is because there are 2 main sources of error: the piece not being placed properly or issues with image quality. 2 extra tries would help mitigate these issues.

We want the user to retry their move at most 2 times before we detect it with 99% accuracy. This means that in total the user gets 3 tries. The reason for this is that there are 2 main situations in which the CV fails. The first is if there is a shadow or lighting issue with the board. The second is if a piece covers a squares boundary. Because there are 2 failure points, we provide 2 extra tries for the user to ensure these are no longer issues.

The way this is tested is the same as detection accuracy described below.

2.1.2 Detection Accuracy After 3 Moves

The target detection accuracy is 99%. This accuracy is with the assumption that the user will let us know if the move we detected is incorrect. We aim for 99% and not 100% because we are limiting the number of retries the user has to make for the sake of convenience, so there may be situations in which we do not get the move right even with the human in the loop. The reason we require this high accuracy is because our entire project is dependent on the detection being highly accurate. If we detect the move wrong, our illegal move detection will not be meaningful. The AI move generator will also be provided the wrong state of the board and generate a nonsensical move. All in all, the entire user experience rests on an extremely high detection accuracy.

While 99% may be too high of an accuracy given that we are relying on canny edge detection which is only 91.56% accurate, the retry mechanism helps in case there is noise in the picture. Additionally, the accuracy of all the edges detected does not matter for our case as we are only use the edge detection algorithm to determine the grid of the chessboard. We will apply the Hough transform to the edges detected to get the horizontal and vertical lines of the grid. We will essentially ignore all the edges other than the ones making up the grid. The edges making up the grid are simple edges, which should further improve the accuracy. Moreover, we use a neural network [3] to crop out the chess board to ensure there is no interference from background elements.

To verify this metric, moves are made for varying pieces, during varying stages of the game. Because exhaustive testing is not possible due to the number of possible moves in chess, we cover a diverse set of moves to ensure the computer vision works:

The test plan is to simulate a real game by providing configurations from the beginning, middle and end of the game. Then, we simply make moves at most 3 times to get the right detection. If the detection is still wrong after the final try, it is counted as an error.

2.1.3 Detection Accuracy after One Move

The idea and testing of this metric is exactly the same as the previous one, except here we stop if we get the move wrong on the first try. The reason this is a metric is because to get a 99% accuracy in 3 tries, the accuracy of each try is limited by the equation:

$$1 - ((1 - x) * (1 - x) * (1 - x)) \geq 0.99 \tag{2}$$

So,

$$x \geq 0.21 \tag{3}$$

This accuracy is very low because it rests on the assumption that the trials are independent. However, we use it as a theoretical bound which we easily exceed.

2.1.4 Sensitivity to Deviation

This is a measure of how much a piece can shift within a square and still not trigger the change detection to detect a change. It is measured as the maximum deviation that will go undetected as a change as long as the piece is in the square. This is required because the user's hand may nudge the piece or the user might accidentally touch it. The most deviation that can occur in one direction that should not be detected as a change is if the piece is at the very bottom of the square. In that case, it can be moved to the top of the square. This is depicted in Figure 1. Each square in our board is 1.875 inches and the average diameter of the piece is 0.8 inches. This means that if the deviation in the vertical direction is y, the maximum deviation that should go unnoticed is bound by:

$$0.8 + y \leq 1.875 \tag{4}$$

This means that

$$y \leq 1.075 \tag{5}$$

So, we require our deviation tolerance to be greater than 1.075 inches in the case that the piece is at the end of the square. We only need to check this case because this is the maximum deviation that is not actually a change. This was tested using pieces of different radii and moving them from the bottom to the top of the square and checking if the value of the change was above the threshold.

2.1.5 Detection Time

This requirement stems from Equation (1) that constrains the total time that the CV, illegal move detection and game playing AI can take. Since the AI takes at most 10 seconds and the illegal move detection at most 2 seconds, it leaves us with 58 seconds for detection time. The way this is tested is similar to the tests in 2.1.2, except we time each run through the move detection pipeline. For each set of up to 3 tries, the maximum time across all tries is taken as the time of the detection.

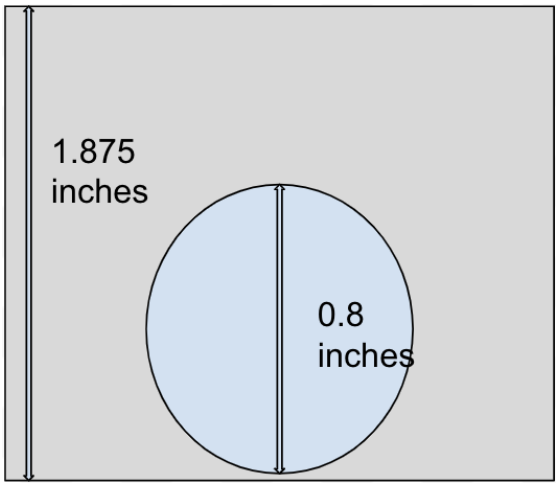

Figure 1: A piece at the bottom of the square

2.2 Game playing AI

We will be integrating an existing Chess AI engine to our project. There are multiple options available, but our major requirements for Chess AI engine was
1. Available for public use (open source)
2. Fast response time (<10s)
3. Offers varying levels of difficulty (optional)
Our final decision stood on Stockfish engine, which offers 8 varying levels of difficulty, response time ranging from 50ms to 400 ms, and publicly posted on github as an open source.

2.3 LED

Given coordinate pairs from the game software, the LEDs corresponding to the coordinates should light up. Additionally, to correctly address the winner of the game, the LEDs should be able to light up in at least two different colors. Based on the requirements, we need 64 RGB individually addressable LEDs for each square of the chessboard. To verify the correctness of LEDs, we will visually inspect that the appropriate LEDs light up given a set of random coordinate pairs.

3 ARCHITECTURE OVERVIEW

Figure 2 shows a block diagram of the architecture. Figure 3 is a more detailed user interaction diagram of the same.

The player will play on a physical chessboard that has an 8x8 LED matrix installed under it. A camera will be mounted directly above the board. The Stockfish engine is initialised when the user is ready to begin the game. When it is the player's turn, they will make a move by moving their piece on the chessboard. Once they make their move, they will press a button mounted on the chessboard. This will signal the camera to take an image of the board. The image passes as an input to the CV pipeline implemented on the Raspberry Pi.

The CV thread determines the user's move and the Raspberry Pi lights up the LEDs corresponding to the initial and final square. In case the CV was not able to detect the move, the Raspberry Pi lights up all LEDs red to tell the user to retry the move twice. After lighting up the moves, the system waits 10 seconds for the user to click a button if the move was incorrect. If the user does not click a button, the move is assumed to be correctly detected and the move detector implemented on the Raspberry Pi passes control to a legal move detector.

The legal move detector gets the coordinates of the current move and the current state of the board. The state is maintained as an 8 by 8 matrix where each element represents a chess piece type and the color. For example, WP is the white pawn. The legal move detector checks if the move is valid. If not, the Raspberry Pi makes all LEDs light up in red and the game is over. Otherwise, the state of the board is updated internally. This is done by simply making the move that the player made. The new move is then fed to the Stockfish chess engine which comes up with the next move. The internal state of the chess board is again updated. The move is displayed and the user physically makes the move and clicks the button. The purpose of this button click after the AI's move is to ensure that the change for the next user move is detected with reference to the board after the AI's move has been made. Once this is done, all LED's light up and the user makes their next move. The game continues.

There were 2 main changes from the design report to the overall system: 1. The retry mechanism, which the user can use to have the system redetect wrong moves. This was to mitigate CV risks. 2. The user is now required to push the button after making the AI move. The reason for this is so that we can use direct change detection between the picture at t=T-1 and t=T instead of having to do change detection between t=T-2 and t=T and checking to ensure the AI move isn't detected as the user's move.

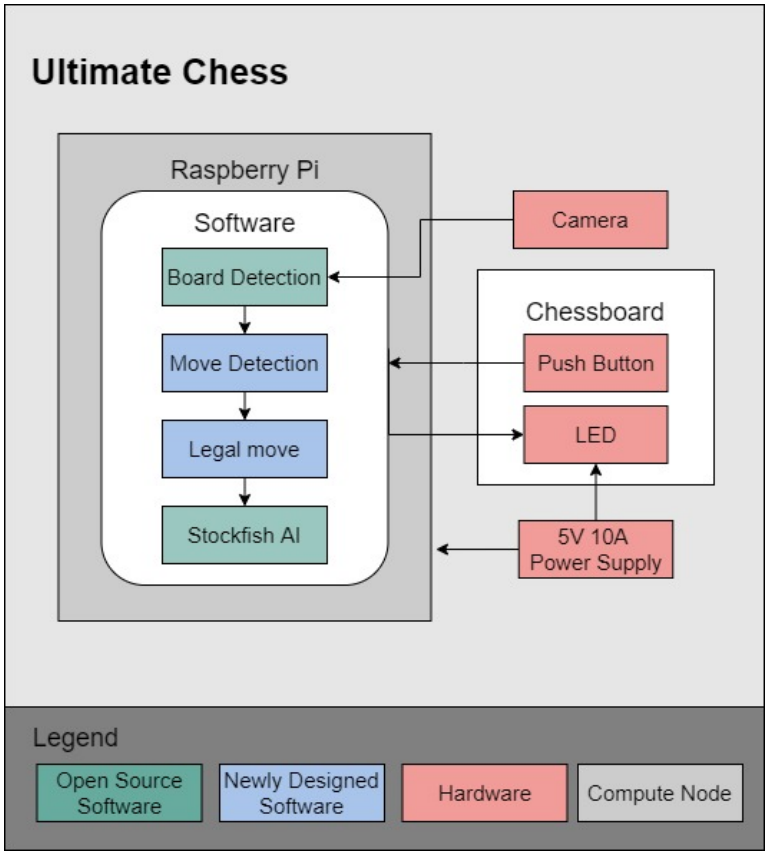

Figure 2: Block diagram

4 DESIGN TRADE STUDIES

4.1 Stockfish Chess AI Engine

We are integrating already existing chess AI 'Stockfish' into our project. Currently there are numerous options available for chess AI engines, but Stockfish has suited our needs the best. Our major requirements for the chess engine were availability to public use, and fast response time to decrease latency. Our optional requirement was offering various levels of difficulty to meet the needs of a wider range of users. Considering these requirements, open source Stockfish AI with 8 varying levels and response time ranging from 50 to 400ms was the optimal option for us.

4.2 WS2812B LED Strip

An 8x8 LED matrix display will be constructed and installed under the chessboard. The 64 LEDS would need to be individually programmable to correctly display the AI's move. To address the winner of the game, the LEDs should be able to light up in at least two different colors.

We chose to use an LED strip over individual LEDs because LED strips are generally cheaper. Additionally, it is less circuit work for us to use the LED strips since the LEDs come in wired together.

The main factor that determines the cost of LED strips is how well it handles the loss of color accuracy due to voltage drop. 5V WS2812B is the cheapest and most common type of LED strip. For WS2812B, voltage drop happens after 2.5m or 150 LEDs. Since we are using a relatively small number of LEDs, only 64 LEDs for each square, voltage drop will not be an issue for our case. Therefore, we are able to use the most cost effective 5V WS2812B LED strip for the project.

4.3 Board LED Integration

In our project, AI moves are communicated to the player by lighting up the appropriate LEDs on the board. There were two possible ways to achieve this. The first way was to buy a standard chessboard set and drill small holes in each of the squares to allow LED lights to go through. The second way was to create our own custom chessboard using translucent acrylic sheets allowing the LED lights to pass through. We chose to go with the second approach because the drilled holes in the first approach may affect the performance of computer vision when detecting the squares and pieces. Moreover, there may be a case where the chess pieces block the drilled holes causing the players not able to see which LEDs have lit up.

Using the second approach means that we have to construct our own chessboard. The top of the board is made of translucent acrylic sheets which are non see-through but still allow light to pass through. White and green acrylic

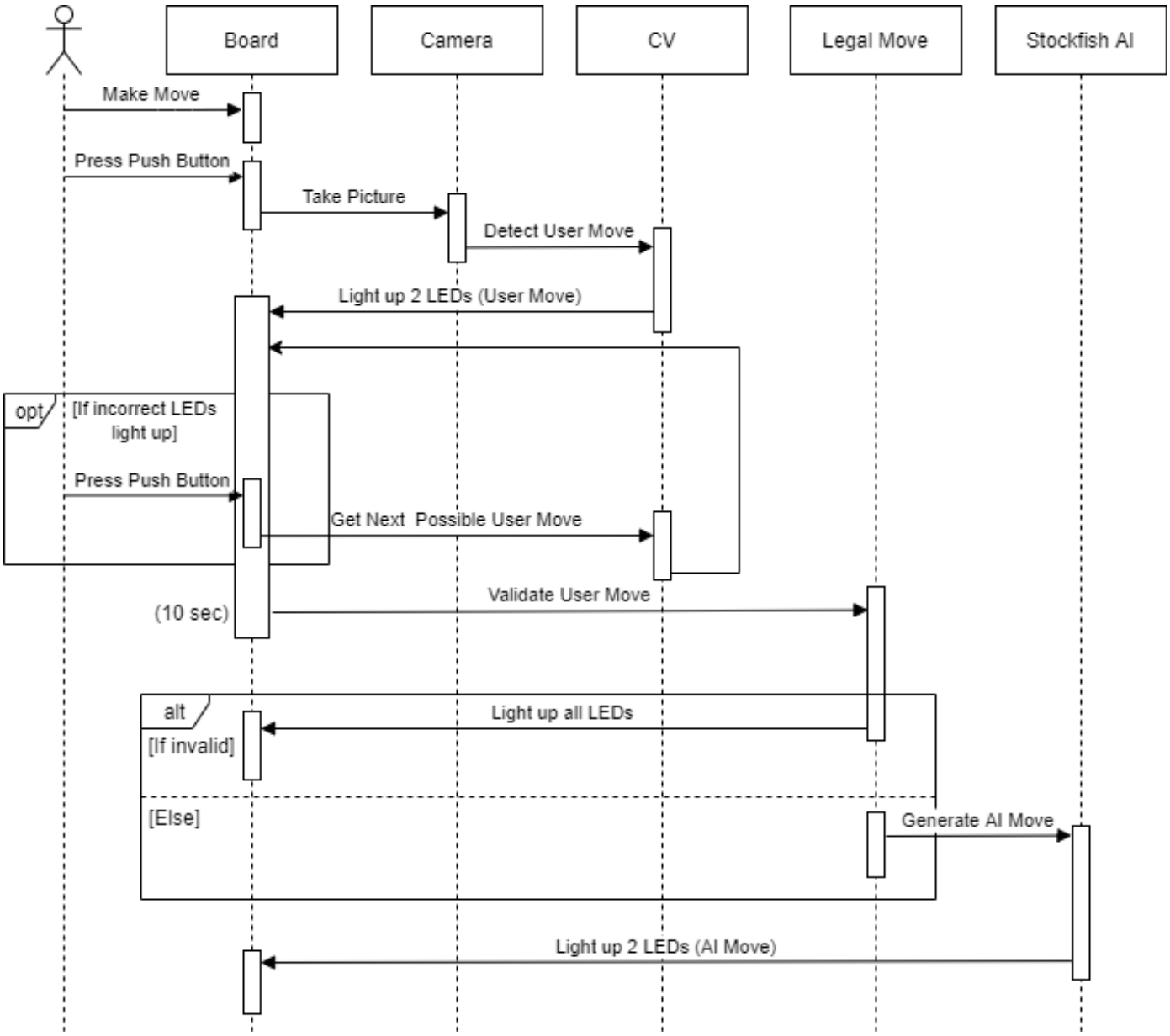

Figure 3: User interaction diagram

Figure 4: Image of the final board (left) and LED matrix (right)

sheets are laser cut to 64 2 by 2 inch squares. The squares are welded into one piece using acrylic weld-on. An 8x8 LED matrix is installed under the chessboard. LEDs are separated from each other using grids which are laser cut from cardboard. Figure 4 shows our completed board and LED matrix.

4.4 Computer Vision for Move Detection

4.4.1 Piece Detection v.s. Change Detection

There are 2 main approaches to determining the move the user made. The first is detecting every piece on the board and figuring out which piece changed position. This provides both the move information and information about which piece is at which square directly. The other approach, i.e. the one we employed, is simply detecting changes in the board and treating pieces as nothing but a set of edges. The edges themselves do not give us any significant information about the piece and we do not try to identify the piece based off of what it looks like. Instead, we identify pieces by tracking their positions through time. Since we know the position of each piece at t=0 (i.e. before any move is made), we are able to determine the new position of a particular piece at t=T simply by combining the position of the piece at t=T-1 with the move made by the human and the AI at t=T to determine the new position of the piece.

There are 3 benefits to this approach:

1. Invariance to the exact shape of the piece: Since we do not rely on the shape of the piece to determine its position, we are able to support different kinds of piece shapes. This is useful because it lets the user play with their own chess pieces instead of us having to provide them. This cuts costs for us because we do not have to provide the user chess pieces. This also makes the set less expensive if the user already owns a regular chess set.

2. Only one top view required: Since we do not need to detect the identity of the piece using its shape, we can simply mount a camera on the top of the chessboard to get a top-down view of it. This is less intrusive than, for example, having to mount 3 or 4 cameras all around the chessboard to ensure that we have enough views of a piece to be able to get an unoccluded image that lets us detect its shape.

3. Less computation: Since we are not using the edge profile of a piece to figure out which piece it is, we save on computation that helps us stay within the time constraints described in equation (1).

4.4.2 Edge Detection Algorithm

The edge detection algorithm is what gives us the edges of the chessboard, the edges of the squares, and the edges of the piece inside the square. Here, we want extremely high accuracy, even if it comes at the cost of time complexity. This is because the results of edge detection feed into the entire move detection pipeline, so we need as high an accuracy as we can get here as the mistakes cannot be made up for later in the pipeline. Therefore, we picked the Canny edge detector which has the highest accuracy.

A table comparing the edge detectors is shown in Figure 5 [3]. As seen in the figure, the Canny operator has the least sensitivity to noise and the least number of false edges. Since we are dealing with real world images, a low noise sensitivity is very important to us. Additionally, since we use edges to discretize the chess board into squares, we must not have false edges as that would lead to the wrong grid being formed which will be disastrous for the entire move detection system.

S.NO	Opertor	Complexity		Noise sensitivity	False Edges
		Time	Space		
1	Sobel	lower	high	Less Sensitivity	More
2	Canny	high	high	Least Sensitivity	Least
3	Robert	high	high	Sensitivity	More
4	Prewit	low	lower	Least Sensitivity	More
5	Laplacian of Gausian	low	least	Least Sensitivity	More
6	Zero crossing	low	less	Least Sensitivity	More

Figure 5: Comparison of edge detectors [2]

4.5 Speed vs Accuracy using a Neural Network

A pre-trained neural network ([2]) is used to crop out the chessboard from an image clicked using the webcam.This adds an average latency of 38 seconds. However, it also improves the accuracy from 65% to 94%. The reason for this is that without cropping out the chessboard, the Hough transform cannot successfully differentiate the lines in the image from the lines being present in the surroundings. Moreover, because the neural network straightens the image, we do not need the user to ensure that the board is completely straight before we run our move detection. We prioritized accuracy in this trade-off because the accuracy of move detection is very important for our project.

4.6 Turn Based v.s. Continuous Game

This was a smoothness v.s. accuracy and time trade-off. The player is required to click a button to tell the system that they are done with making their move. This is in contrast to the camera capturing a continuous video stream and the CV checking to see whether or not the user is finished with their move. While the latter approach is smoother for the user, it creates limits both our accuracy and speed. The reason for this is that the CV now has to detect whether or not the move has been completed. This adds an average time of 55 seconds to the move. Moreover, it reduced the accuracy by 12.5% if we use standard hand detection methods to ensure the users hand is not in the image when we start detecting the move.

4.7 Retries vs No Retries (Smoothness vs Accuracy)

This was a smoothness v.s. accuracy and time trade-off. The player is required to click a button to tell the system that they are done with making their move. This is in contrast to the camera capturing a continuous video stream and the CV checking to see whether or not the user is finished with their move. While the latter approach is smoother for the user, it creates limits both our accuracy and speed. The reason for this is that the CV now has to detect whether or not the move has been completed. This adds an average time of 55 seconds to the move. Moreover, it reduced the accuracy by 12.5% if we use standard hand detection methods to ensure the users hand is not in the image when we start detecting the move.

5 SYSTEM DESCRIPTION

5.1 Move Detection

Move detection is the system that takes as input the webcam image and ultimately outputs the coordinates of the detected move.

5.1.1 Square Formation

Figure 8 shows how we get from the webcam image to the squares at t=T and t=T-1 that can be passed to the change detection algorithm. First, the images are cropped using a pre-trained neural network that focuses on the chessboard. Then, the images are blurred to remove high frequency noise. Canny edge detection, a method of detecting edges while suppressing noise, is applied. The Canny edge detection was used because of its low sensitivity to noise compared to other methods as seen in figure 8. Using a standard gradient based approach detected the correct lines only 62% of the time as compared to Canny which does it over 98% of the time in our situation. The problem with a gradient based approach comes from the fact that the gradient in our board is often not clear due to shadows or lighting differences. A Hough transform returns the horizontal and vertical lines in the edges. These lines are then rectified to ensure that they are equidistant and any noisy lines are removed. This is done by computing the median distance between the lines and filtering out lines that are more than 1 standard deviation away from either the line

Figure 6: Square formation

before it or the line after it. Then, the intersecting squares are cropped. After this, the squares for the previous and current image are sent to the change detection algorithm. An example image is shown in figure 6.

5.1.2 Change Detection

After each square on the board was cropped, frame difference algorithm of background subtraction was used to detect the move that user had made. The foreground frame is set as a collection of squares of a most recent board state at time = T, and a board state before user has made a move at time = T - 1 was set as a background frame. Our change detection algorithm first sets a bounding circle on each cropped square to reduce the error from any external source, such as inconsistent reflection on the board or lighting. The bounding circle further reduces the error rate by concentrating the pixels to a piece location. After a bounding circle is set for each square, we then cumulatively add a difference value for each pixel per cropped square. The difference value is computed by absolute difference of R value from the RGB code of a pixel. We only used 'red' value to compute a difference value because we used red pieces for user's piece set. Since we are trying to derive a change in location of a red piece, extracting R value from RGB instead of using the entire RGB data set increased our accuracy rate. Finally, we derive a coordinate of a square where the change has been made by setting a threshold. Two squares where a change has been made is going to have a difference value over a threshold, and any other squares will have a difference value under a threshold. We then use the derived coordinates to update our board state based on the user's move.

5.2 LED with Raspberry Pi

The WS2812B RGB LED strip is controlled by the Raspberry Pi using the rpi_ws281x library, which is also known as Neopixel library. In addition, a 5V power supply is required. The single data line of the LED strip is connected to the GPIO pin on the Raspberry Pi. The ground of the power supply is connected to both the ground wire from the strip and the ground pin on the Raspberry Pi. The 5V output of the power supply is connected to the 5V voltage wire from the strip but not to the voltage pin on the Pi. The schematic connection between the LED strip, power supply, and RPi is shown in Figure 7 [1].

6 TEST & VALIDATION

6.1 Results for Accuracy and Latency of Move Detection Specification

There are 2 requirements for accuracy of move detection: accuracy without retries and accuracy with retries. The testing process was the same for both of them, except that in the with retry case the move was considered accurately detected if any one of the 3 tries got it correct.

For each test, we detect both the board detection and the move detection accuracy. The board detection accuracy is the accuracy of partitioning the board into squares. The move detection accuracy is the full accuracy, from the beginning with the original picture to the end where Our testing process involved mass tests and in game tests.

Our testing process involved mass tests and in game tests.

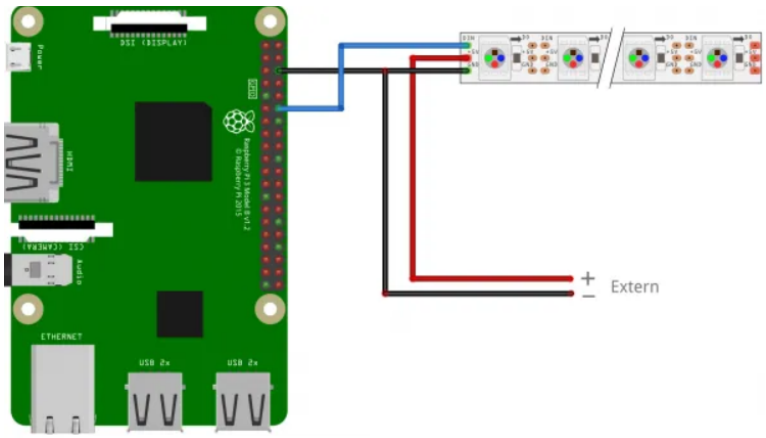

Figure 7: LED schematic [1]

6.1.1 Mass Testing

In mass testing, we changed positions of multiple pieces in one turn. This is not a legal chess move, but it allowed us to check the accuracy of several change detections with one click of the camera instead of clicking pictures for every move. This made testing more efficient. The Our results for mass testing are as in table 1. BD represents board detection accuracy and MD represents move detection accuracy. NR represents no retry and R represents retries are allowed. Since we are only detecting red piece moves, only red pieces are tested. In the first test, all red pieces are moved forward by 1 step. So, each iteration of the test contains 16 moves and we perform 3 iterations. We get a 100% accuracy across the row here. In the second test, we test capturing of a small piece by a big piece. Blue pieces except the king are replaced with a bigger red piece. Only one piece of each piece type is replaced, and this is varied between iterations. The king cannot be replaced because it is bigger than all pieces. So, 5 moves are made in each iteration. In one of the iterations, the board detection failed and so our board detection and move detection accuracies are both low. However, retries bring the board detection to 100% and the move detection to 93%. In the third test, we test capturing of a piece by the same piece type. We 86% accuracy for move detection without retries, but this goes up to 100% with retries.

In all, the board detection accuracy is accurate both with and without retries. However, retries make the move detection accuracy go from 90% to 98% which is very close to our requirements of 99%. The latency is 47.45 seconds which meets our requirement of being under 60 seconds.

6.1.2 Piece-wise Tests In-game

These tests were performed during the game for each piece. The game states were chosen uniformly between initial states (first 1-20 moves), middle states (first 20-30 moves) and final states (30-40 moves) by playing 2 AIs against each other and finding sample game states for each stage of the game. The reason for this is to ensure CV doesn't fail due to game setting or density of the board. The accuracy for the Bishop is the most problematic, and that is because it is thinner that the other pieces and has a more pointed tip. This limits the accuracy of our detection for the Bishop because we are working with top view images.

6.2 Results for Sensitivity of Move Detection Specification

The sensitivity tests were performed 5 times for each piece. We did not require more tests because the values were stable for each piece. In each test, we mass tested all pieces. Each piece was moved from the bottom of the square to the top of the square, and it was checked whether the change value was above the threshold or not. If it wasn't, this was counted as a success. The maximum deviation at which we failed at each piece was recorded. The results are in table 3. This requirement is a limitation for us. The most deviation we were able to achieve was 0.

We were only able to support deviations up to 0.91 in, instead of the requirement of 1.075 in. This shows that our Computer Vision is more sensitive than we would like. We were able to achieve 0.91 inches for Queen, but our lowest was Bishop at 0.43 in.

6.3 Results for LED Accuracy and Latency

In order to achieve our latency requirement, we needed to ensure that the time it takes to change the state of the LEDs is minimal compared to the overall latency. Therefore, we set the LED code execution time to be less than 100ms. The timing of the LED code execution consists of the time necessary to parse the algebraic chess notation into the LED index and lighting up the corresponding LEDs. The average execution time over 30 tests was around 28ms, which as expected is insignificant to the overall latency. We also expected 100% correctness of LED behavior. We performed 30 visual tests. These tests were done by vi-

Table 1: Mass testing

Test type	BD(NR)	MD(NR)	BD(R)	MD(R)	T(NR) sec
Move all forward	3/3	48/48	3/3	48/48	46
Replace blue (except king) with bigger red	2/3	8/15	3/3	14/15	44.5
Replace blue (except pawn) with smaller red	3/3	15/15	3/3	15/15	51.3
Replace blue with same red	3/3	13/15	3/3	15/15	42
Total	11/12	84/93	12/12	92/93	

Table 2: Piece wise in-game tests

Piece type	BD(NR)	MD(NR)	BD(R)	MD(R)
King	6/6	6/6	6/6	6/6
Queen	6/6	5/6	6/6	6/6
Bishop	6/6	4/6	6/6	5/6
Knight	6/6	6/6	6/6	6/6
Rook	6/6	6/6	6/6	6/6
Pawn	5/6	5/6	6/6	5/6

sually confirming that the correct LEDs light up given a move with standard chess notation and color. 30 out of 30 tests passed, so the LEDs behave as expected.

7 PROJECT MANAGEMENT

7.1 Schedule

The major requirement of our project is to be able to detect user's moves through CV image processing. Thus, after finishing the construction of the physical chess board, the first several weeks of the schedule was dedicated to constructing our CV algorithm and testing it. As soon as we concluded that our CV algorithm correctly detects moves, we began integrating Stockfish engine and valid / invalid move logic into our project. While working on final integration, we also focused on testing our project's functionality and its responses to possible edge cases. Our detailed schedule is in Figure 9 at the end of the document.

7.2 Team Member Responsibilities

Demi worked on constructing the physical board and hardware assembly including communication between the board and the Raspberry Pi and camera, LED matrix construction, and the integration of user input push button. Yoorae worked on implementing the chess valid logic to check for valid/invalid user moves and also integrated the chess AI with the game. Anoushka mainly worked on CV image processing. Once the valid chess logic was done, Yoorae and Anoushka worked together on background subtraction algorithms for player move detection. Once the board was constructed and LEDs were tested, Demi worked on the overall integration between the three main portions of the system: board, CV, AI.

7.3 Budget

Our group spent a total of $184.36 for building our project. The details of purchased components and their cost is attached in table 4 below.

7.4 Risk Management

Most of our risks this semester were related to Computer Vision. This is because there are a lot of real world factors that impact its accuracy: movement of the board, glare etc. We scheduled a lot of time for CV and were prepared for several iterations. We began testing CV performance early before we got our web cam using images from our phones. We also ordered our web cam and stand early so we could change it if it didn't work. We did have to change one of our stands because it wasn't stable. To further mitigate the risk from CV, we decided to use a neural network to crop the chess board out before we begin the process of square partitioning. While this did improve our accuracy, we didn't have a lot of resources to help with this so setting it up and using it took a significant amount of time. We used open source resources to combat this. Initially, our CV wasn't good enough even after cropping and change detection was very inaccurate. We had planned for this in our design report, and immediately got pieces of Red and Blue color instead of the standard black and white pieces. This fixed the problem we were having because the pieces were no longer the same color as the squares. To further mitigate risks, we introduced a new design element: the push button. This allowed the user to tell the system if the move was incorrectly detected so the system could re-detect the move.

Table 3: Sensitivity tests

Piece type	Max deviation (in)
King	0.86 in
Queen	0.91 in
Bishop	0.43 in
Knight	0.67 in
Rook	0.86 in
Pawn	0.83 in

Table 4: Bill of materials

Description	Model #	Manufacturer	Quantity	Cost @	Total
Chess board and pieces set	N/A	Chess Armory	1	$28.99	$28.99
Logitech C270 Webcam	C270	Logitech	1	$26.92	$26.92
Raspberry Pi 4 (Capstone Inventory)	B	LABISTS	1	$0	$0
LED Strip	WS2812B	BTF-LIGHTING	1	$22.88	$22.88
5V 10A Power Adapter	N/A	ALITOVE	1	$23.99	$23.99
Cast Acrylic Sheet White	N/A	McMaster	1	$31.91	$31.91
Cast Acrylic Sheet Green	N/A	McMaster	1	$31.91	$31.91
Weld-on 4	N/A	WELD-ON	1	$17.85	$17.85
					$184.36

8 ETHICAL ISSUE AND USE CASE

The current COVID-19 pandemic has excluded the elderly and others who are uncomfortable with modern technology from society, since social interactions based on physical contact have immensely decreased. Even before COVID-19, exclusion of older adults from fastly developing technology has been an ethical issue. Our project aims to provide leisure to anyone who is experiencing such exclusion from modern society by developing a system that allows users to play a chess game without any physical contact with another individual and any software-based interaction.

9 RELATED WORK

Our project was initially inspired by the smart chess board 'Square off' when formulating ideas of the project. 'Square off' is an automated chess board that users can make a move on a physical chess board, and AI will respond with an automated move of pieces through circuited magnets inside the board. To reduce the complexity of the hardware model, we pivoted to the idea of updating user's moves by image processing from a top view camera and displaying AI's move on LEDs of the board. The group 'Chess Teacher' from last semester's ECE Design Experience course had a similar project with us. They detect user's moves through image processing of images taken from a top view camera, and display the AI's moves through their front end UI. The biggest difference between our project and 'Chess Teacher' will be the display of AI's moves. Our major goal is to get rid of any software components from user experience to remove exclusions from any users, such as elderly, who are having difficulties with software components. The users of ultimate chess will not be required any interaction with front-end UIs, and every communication in the game will be fully physically visible.

10 SUMMARY

So far, our team managed to achieve our MVP with relatively small delays from our original schedule. Our CV's change detection maintained viable accuracy rate, and user can still finish a game of chess even through there is an error from CV through a 'retry' option. There biggest limitation of our project still exists; the latency for processing user's move is high. It takes about 45 seconds to process a single move, and the wait time is pretty long from a user's perspective to play a smooth game. If we had time to further improve our project, our group would have focused on decreasing our latency.

10.1 Future Work

The biggest limitation of our project is a CV latency bottleneck. It takes about 45 seconds to process each move that user makes, and it creates an uncomfortable delay from a user's perspective. If we were given more time on the project, we would focus on decreasing the latency as much as possible. The first possible approach is getting rid of neural network to process cropping of the board. Board needs to be cropped to accurately process our edge detection. One way to get rid of neural network from our process would be to reconstruct our board and camera set up so that the camera will always shoot from a consistent

distance with consistent angle. Then, we would be able to manually crop the board without neural network by hard coding the coordinates of four edges of the board.

References

[1] *Connect and control WS2812 RGB led strips via Raspberry Pi*. 2021. URL: https://tutorials-raspberrypi.com/connect-control-raspberry-pi-ws2812-rgb-led-strips.

[2] Maciej A. Czyzewski, Artur Laskowski, and Szymon Wasik. *Chessboard and chess piece recognition with the support of neural networks*. 2020. arXiv: 1708.03898 [cs.CV].

[3] S.K. Katiyar. "Comparitive analysis of common edge detection techniques in the context of object extraction". In: *IEEE TGRS Vol.50 no.11* (2012), pp. 77–78.

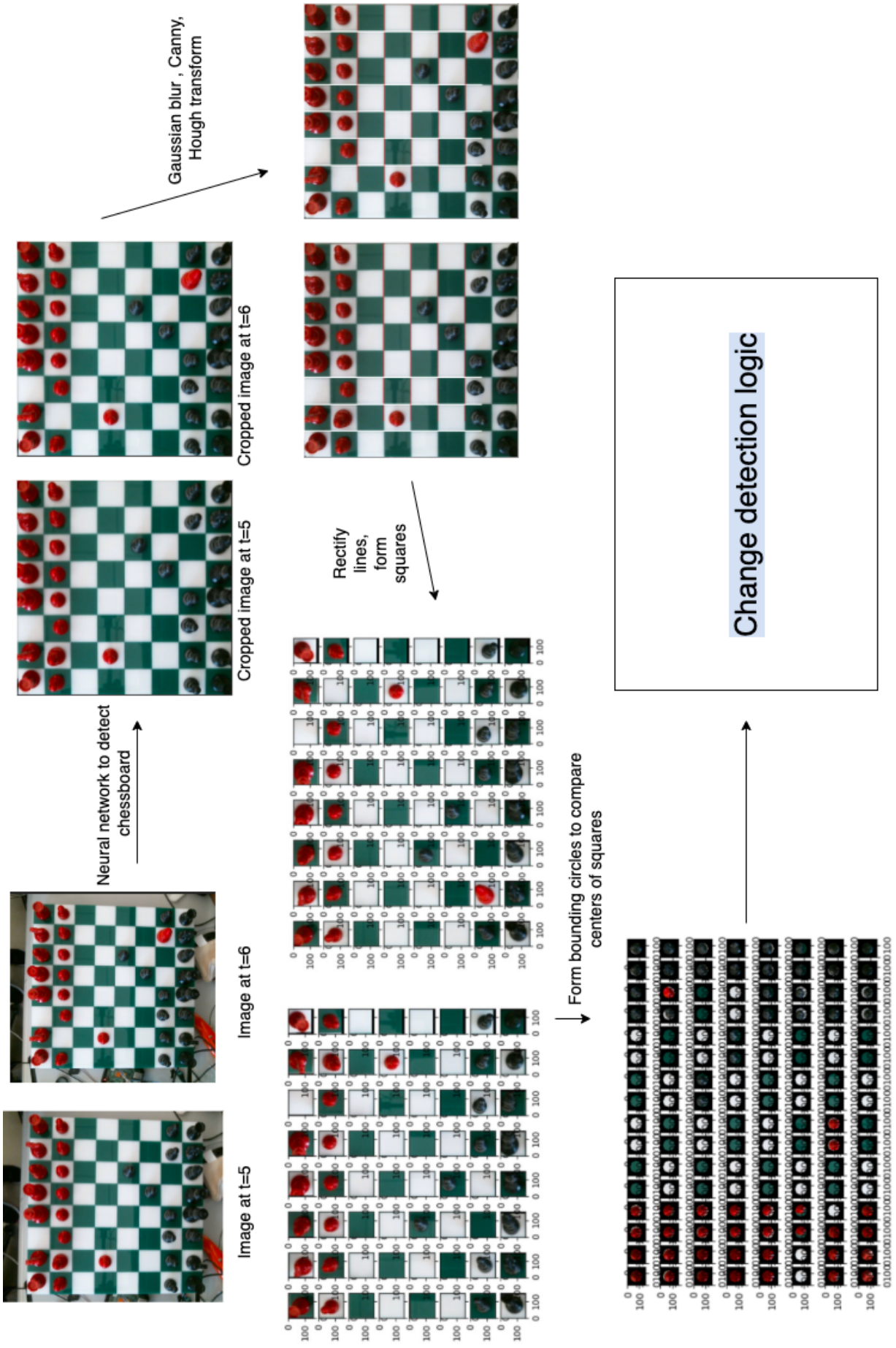

Figure 8: CV flow chart

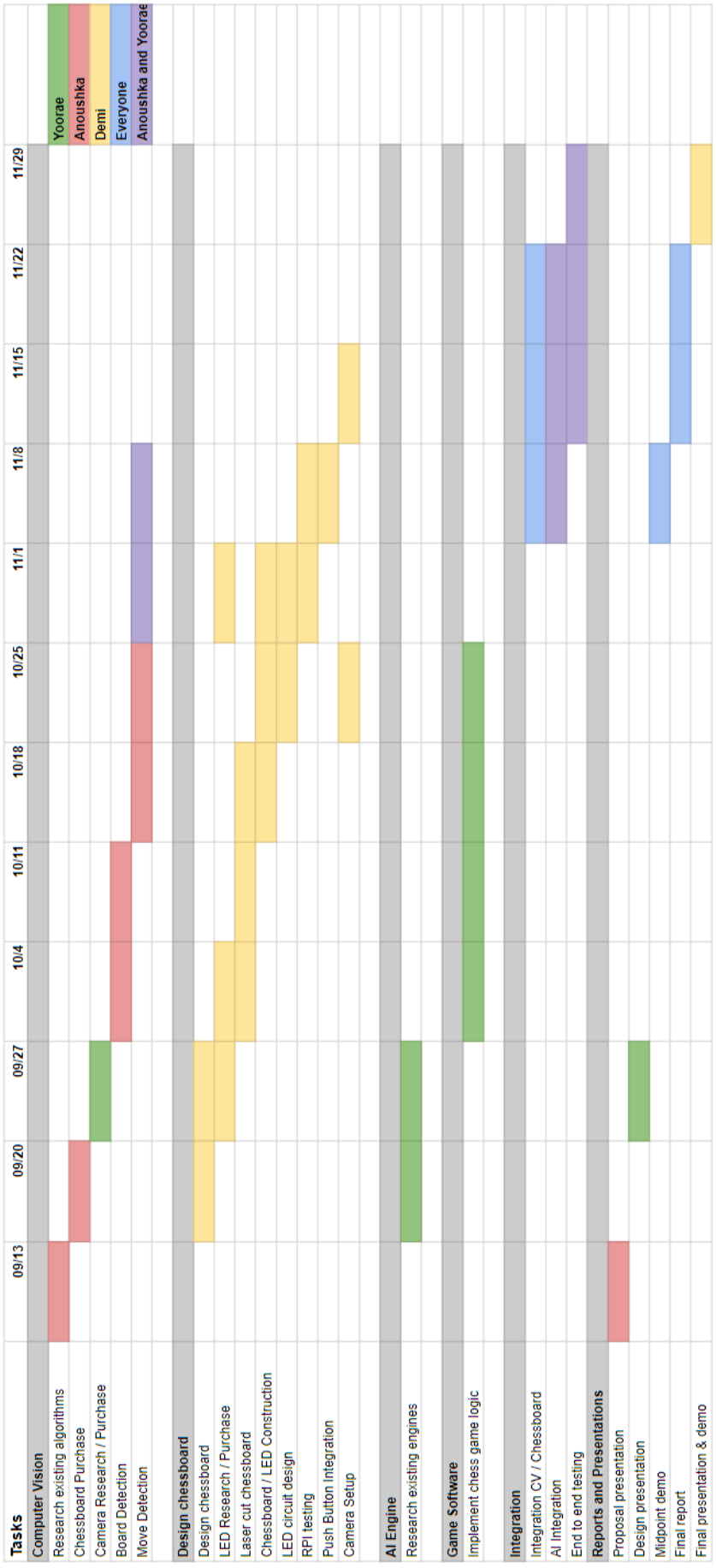

Figure 9: Schedule